

采用 CUDA 技术的视频图像块效应评价算法

魏晓娟, 陈庆奎, 王海峰

(上海理工大学 光电信息与计算机工程学院, 上海 200093)

摘 要: 为了对视频图像质量进行实时评价, 在分析现有块效应评价算法基础之上, 提出了在流处理器并行环境下视频图像块效应评价算法, 采用 CUDA 技术实现该算法程序. 实验结果表明, 在 NV Geforce GTX280 上实现的并行算法比 CPU 上的串行算法提高了近 30 倍的效率. 证明了 CUDA 技术在数字图像处理应用中具有极大的潜力, 尤其是计算密集型任务处理.

关键词: 流处理器; CUDA 技术; 块效应评价; 视频图像

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2010)08-0256-05

An Evaluation Algorithm of Blocking Artifacts in Video Images with CUDA Technology

WEI Xiao-juan, CHEN Qing-kui, WANG Hai-feng

(School of Optical-Electrical and Computer Engineering, University of Shanghai for
Science and Technology, Shanghai 200093, China)

Abstract: In order to give an efficient evaluation of the quality of video images, based on the analysis of the existing algorithms of evaluation, this article proposes an realization of these algorithms under the Stream Processor Parallel environment and program with the technology of CUDA. The result shows that the speed of parallel computing in the NV Geforce GTX280 is about 30-time of the serial algorithm running in the CPU. This proves that CUDA technology has great potential in the digital image processing and particularly the CPU-intensive mission processing.

Key words: stream processor; CUDA technology; evaluation of blocking artifacts; video images

1 引言

目前, 基于块的离散余弦变换(BDCT)已经成为静态和动态图形图像压缩技术的重要组成部分^[1], 包括 JPEG、H. 263 /264、MPEG 等图形图像编码标准均采用了此方法. BDCT 将图像分割为 $N \times N$ 大小的像素块, 这些块各自独立的进行离散余弦变换(DCT)、量化、编码和传送. 由于系数粗量化计算和相邻两个像素块之间独立的处理过程, 导致在像素块边缘产生了人为的像素值跳跃, 反映在解码后的图像上就会呈现出一种阶梯状的视觉效果,

这就是所谓的“块效应”. 块效应已经成为 BDCT 技术中最大的缺陷.

为解决这一问题, 学术界现已提出多种块效应评价算法和去块效应的方法^[2-5]. 但文献[4]和文献[5]中提到的块效应评价算法均没有结合人眼识别系统的特征, 只能用在块效应、模糊度等空间特征的测量上^[6].

本研究项目是关于分布式、实时 3G 质量监测系统, 需要处理大规模的实时视频流, 一般 CPU 难以承受.

文中采用并行计算的串行化块效应算法—

收稿日期: 2010-05-13; 修回日期: 2010-06-12

基金项目: 国家自然科学基金项目(60970012); 上海信息技术领域重点科技攻关项目(09511501000); 上海重点科技项目(09220502800); 上海市重点学科建设项目(S30501)

HVSBM^[7]. 利用 NVIDIA 公司推出的 CUDA 平台, 开发视频图像块效应评价算法的并程序, 并通过优化达到更好的效果. 实验结果表明, 和传统的 CPU 串行程序进行对比, 视频图像块效应评价算法的并程序的运行速度加快近 30 倍.

2 CUDA 编程模型

GPU 作为一种高效的流处理计算单元, 既可以进行复杂的图像处理, 还可以利用其强大的流处理器来进行广泛的应用计算, 因此适合当今对流计算并行计算要求. NVIDIA 公司开发的计算统一设备架构 (Compute Unified Device Architecture, CUDA)^[8], 简化了 GPU 的并行计算的开发流程, 使得人们对 GPU 的应用摆脱了对硬件 API 的依赖^[9-10]. 这种新统一渲染架构的形成, 比其它 GPUs 提供了更大的存储带宽和更高的计算能力, 更有利于通用计算领域.

采用的 GPU 是 NVIDIA GTX280, 其核心频率 602MHz, 具有 30 个多核流处理器 (Streaming Multiprocessors, SMs), 工作频率 1296MHz. 存储器层次结构上, 最大存储带宽为 140.8GB/s. 具有 4 个不同类型的片上存储器: 常量高速缓存 (constant cache)、纹理高速缓存 (texture cache)、寄存器 (registers)、共享内存 (shared memory). 常量高速缓存和纹理高速缓存都是只读的, 可被所有 SPs 共享访问.

CUDA 程序设计遵循着固定的编程模型, 一个 CUDA 并程序分为 CPU 和 GPU 部分: 其中 CPU 端程序称为 Host 端, 只负责调用和控制 GPU 计算模块; GPU 端是真正的并行计算模块, 称为 Device 端. 按照 CUDA 开发文献描述, 其过程大致分为以下几步:

(1) 将数据从外设存储器复制到 Host 的内存中;

(2) 将数据从 Host 内存复制到 Device 的内存中;

(3) 由 Host 端调用 Device 端的 Kernel 并程序来执行并行计算;

(4) Device 端建立子线程, Kernel 并程序开始执行计算, 计算完成后将结果写在 Device 端的内存中;

(5) Device 端的 Kernel 并程序结束后, 返回 Host 端, 并将结果数据从 Device 端转移到 Host 端.

文中设计也遵循此模式.

3 基于人类视觉系统的图像块效应评价算法 HVSBM 介绍

首先, 利用 Sobel 边缘检测算子区别出视频图像上的边缘区域和平滑区域. 设当前视频图像中一个块区域边界上的某个像素点为 $f(x, y)$, 该点上的局部梯度分量定义如式(1)所示:

$$g_x = f(x+1, y-1) - f(x-1, y-1) + 2f(x+1, y) - 2f(x-1, y) + f(x+1, y+1) - f(x-1, y+1)$$

$$g_y = f(x-1, y+1) - f(x-1, y-1) + 2f(x, y+1) - 2f(x, y-1) + f(x+1, y+1) - f(x+1, y-1) \quad (1)$$

在该点 (x, y) 上梯度的角度方向和量级定义如式(2)所示:

$$\theta(x, y) = \tan^{-1} \left\{ \frac{g_x}{g_y} \right\}$$

$$G(x, y) = \sqrt{g_x^2 + g_y^2} \quad (2)$$

对于给定的边缘阈值 T , 当 $G(x, y) \leq T$ 时, 认为像素点 $f(x, y)$ 为非边缘点, 反之则为边缘点.

当 $G(x, y) > T$ 且 $|\theta(x, y)| = \frac{\pi}{2}$ 时, 检测出来的应该是条垂直的边缘, 但也不排除是水平临界的两个块之间的块效应造成的, 所以对此像素点 $f(x, y)$ 做特别处理.

接下来进行块效应的计算. 设非边缘点和边缘点的块效应分别为 D_1 和 D_2 , 定义如式(3)(4)所示:

$$D_1 = (f(x, y) - f(x, y-1))^2 \quad (3)$$

$$D_2 = (f(x, y) - f(\theta(x, y)))^2 \quad (4)$$

其中, $f(\theta(x, y))$ 是像素点 $f(x, y)$ 沿着边缘 $\theta(x, y)$ 方向上的邻接点. 特别情况, 当 $G(x, y) > T$ 且

$|\theta(x, y)| = \frac{\pi}{2}$ 时, 定义如式(5)所示:

$$D_3 = |f(x, y) - f(x, y-1)) - 1/2[(f(x, y-1) - f(x, y-2)) + (f(x, y+1) - f(x, y))]|^2 \quad (5)$$

于是, 得到一个评价像素点 $f(x, y)$ 水平方向上的块效应的结果 D_{HVSBM} 如式(6)所示:

$$D_{HVSBM} = I(G(x, y) \leq T \times D_1 + \alpha(I(G(x, y) > T, |\theta(x, y)| \neq \frac{\pi}{2})) \times D_2 + I(G(x, y) > T, |\theta(x, y)| = \frac{\pi}{2}) \times D_3 \quad (6)$$

其中, $I(*)$ 是示性函数, 当满足条件 $G(x, y) \leq T$ 时为 1, 否则为 0. α 为调节 D_1 和 D_2 的权值参数.

4 基于人类视觉系统的视频图像块效应并行评价算法

4.1 基于 CUDA 的并行化数据结构

为了达到最优的并行化效果,在利用 CUDA 开发并行化程序时,数据结构的合理安排和组织管理成为关键的步骤.在 CUDA 开发环境中,针对本算法有多种模式的存储结构供选择,包括 global memory, shared memory 和 texture memory.考虑到程序的优化和运行效率,本项目采用 global memory 和 shared memory 来装载待处理的视频图像数据.

该算法所涉及到的数据结构向 CUDA 映射后的具体描述(假设 area 为每帧像素点的个数, numframe 为需处理的帧数):

(1) Host 端上的数据

① $h_framestream[area * numframe] = [P_{0,0}, P_{0,1}, \dots, P_{0,area-1}, \dots, P_{numframe-1,0}, P_{numframe-1,1}, \dots, P_{numframe-1,area-1}]$, 存储所有待处理的视频图像数据.

② $h_gx[area * numframe]$ 和 $h_gy[area * numframe]$, 分别存储 Device 端传回来的 d_gx 和 d_gy .

③ $h_Result_Buffer[area * numframe]$, 存储 Device 传回来的 d_HVSMB .

(2) Device 上的数据

① $d_Y[area * numframe]$, 存储视频图像 YUV 文件中 Y 分量.

② $d_gx[area * numframe]$ 和 $d_gy[area * numframe]$, 分别存储经过 Sobel 算子计算后的 g_x 和 g_y 分量.

③ $d_HVSMB[area * numframe]$, 存储经过本并行化算法后的 HVSMB 结果.

4.2 初始装载过程

参照 CUDA 的一般初始装载过程,具体描述为:

(1) CUDA 硬件初始化: $CUT_DEVICE_INIT(argc, argv)$.

(2) 从硬盘读取视频流 yuv 文件中的 Y 分量复制到主存 readyuv(输入层数据).

(3) CUDA 硬件计算存储空间分配: $CUDA_SAFE_CALL(cudaMalloc((void **), size))$.

(4) 视频流装载,方向由主存到设备: $CUDA_SAFE_CALL(cudaMemcpy(dst, src, size, cudaMemcpyHostToDevice))$.

4.3 Kernel 线程设置与流处理器分配

具体线程设置和流处理器的分配情况如下:

为说明简单,假设视频 YVU 文件每帧大小为 $weight \times height = 176 \times 144$, 共有 $frame_num = 300$

帧.

为使 CUDA 的运行效率达到最高,设计每个 Block 共有 $BLOCK_SIZE \times BLOCK_SIZE = 16 \times 16$ 个线程,而每个线程处理一个像素点.则每一帧需要有 $weight/BLOCK_SIZE \times height/BLOCK_SIZE = 176/16 \times 144/16 = 99$ 个 Block. 设计 Grid 为:每行处理一帧,则 Grid 中的一行包含 99 个 Block,共有 $frame_num = 300$ 行.线程设置分配表和示意图分别如表 1 和图 1 所示.

表 1 线程设置分配表

图片帧大小		帧数	Grid 维度(dim3)	
Weight	Height		X	Y
176	144	$frame_num = 300$	$weight/BLOCK_SIZE \times height/BLOCK_SIZE = 176/16 \times 144/16 = 99$	$frame_num = 300$

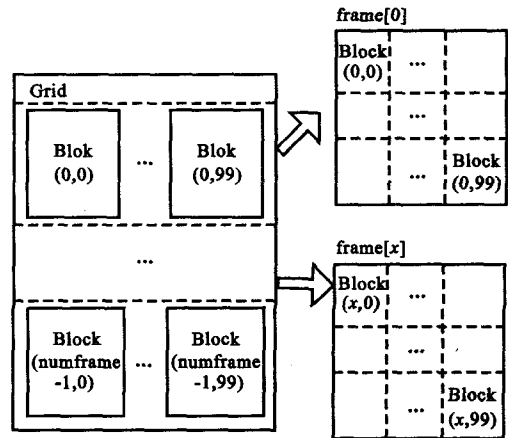


图 1 线程分配示意图

4.4 流处理器上的数据收集

进入 Kernel 并程序后,数据收集发生在 Sobel 边缘检测之前.首先将全局内存中的数据转移到每个 Block 所使用的共享存储器上,将 Device 端上的数据复制到 shared memory 上.数据转移过程中需要注意边界问题,以免发生越界访问,导致失败.

4.5 数据回收到 Host 端

所有线程计算完毕后,将计算结果通过 Host 端控制从 Device 端转移到 Host 端的内存上并输出.

$CUDA_SAFE_CALL(cudaMemcpy(h_Result_Buffer, d_HVSMB, area * numframe * sizeof(float), cudaMemcpyDeviceToHost))$.

5 算法描述

5.1 输入程序运行信息

分别输入待分析的视频流文件,视频帧的宽和高,视频流帧数和程序运行轮数.

5.2 初始装载

详见 4.2 节.

5.3 内核启动算法

算法 Kernel 启动

HandleFrame<<< dimGrid, dimBlock >>>
(d_Y, numframe, d_gx, d_gy, d_HVSBM, width,
height).

5.4 Kernel 并程序算法

(1) 在 Device 初始化每个 block 的共享内存
shared memory:

__shared__ BYTE s_Y[BLOCK_SIZE + 2]
[BLOCK_SIZE + 2];

(2) 将 block 中各个线程对应的 global memory
的数据复制到 shared memory 中:

s_Y[threadIdx.y + 1][threadIdx.x + 1] = d_Y[index_Y];

(3) 计算 Sobel 边缘检测算子,按照第 3 节中式
(1)计算得到 g_x, g_y 和式(2)计算得到量级 G 和角
度 $Atan$;

(4) 按照第 3 节式(6)中量级 G 和阈值 T 相比
较的三种情况,计算出 DHVSBM 值.

5.5 计算结果返回用户区

(1) Kernel 并程序计算结束,CPU 将块效应
分析结果转移到 Host 内存中:

cudaMemcpy(h_Result_Buffer, d_HVSBM,
area * numframe * sizeof(float), cudaMemcpyDe-
viceToHost);

cudaMemcpy(h_gx, d_gx, area * numframe,
cudaMemcpyDeviceToHost);

cudaMemcpy(h_gy, d_gy, area * numframe,
cudaMemcpyDeviceToHost);

(2) 释放 Host 端和 Device 端上所占资源.

6 实验设计与分析

本实验运行环境是 NVIDIA GeForce GTX280
显卡, Intel Core2 E8400 3.0GHz 的 CPU. 编译环境
为 Visual Studio2005 和 CUDA3.0.

(1) 对相同视频流文件和相同的 HVSBM 算法
分别在 CPU 和 GPU 上计算. 计算结果如图 2 所示.

实验结果表明: CPU 和 GPU 上 HVSBM 算法
运行结果基本保持一致, 由此验证了 GPU 并行计
算结果的正确性.

(2) 借助 CUDA 自带的测试时间函数 cutCre-
ateTimer、cutStartTimer 和 cutStopTimer 来统计程

序运行的时间(包括数据从 Host 端转移到 Device 端
的时间, GPU 的计算时间, 以及将结果回传到 Host
端的时间), 并对同一 YVU 文件循环运行了 5 次,
最终求出每次运行的平均时间, 来作为运行效率的
对比标准. 不同大小的 YVU 视频文件的 CPU 运行
时间、GPU 运行时间及加速比如表 2 所示. GPU 和
CPU 的运行结果对比如图 3 所示.

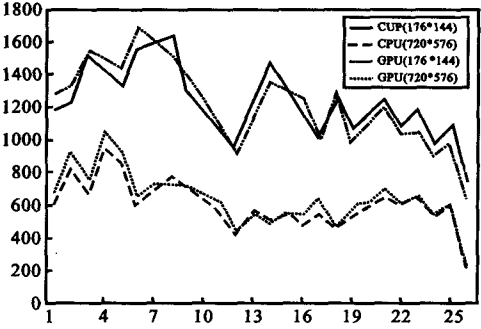


图 2 CPU 和 GPU 运行结果 HVSBM 对比

表 2 CPU、GPU 运行时间及加速比

图片尺寸大小	176 * 144	352 * 288	640 * 480	720 * 576
帧数	300	300	300	300
CPU 运行时间(s)	1.217	4.855	15.236	19.373
GPU 运行时间(s)	0.051	0.197	0.612	0.683
加速比	23.86	24.64	24.9	28.36

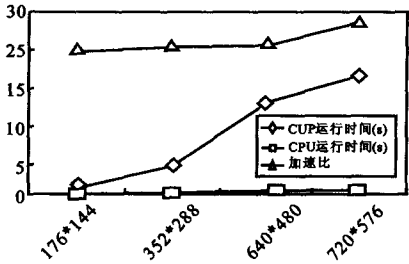


图 3 CPU 和 GPU 实验结果对比

实验结果表明: 利用不同的计算资源, 相同视频
图像块效应评价算法的运行时间是不一样的, GPU 上
的运行效率是 CPU 上运行效率的近 30 倍. 并且随
着 YUV 文件视频数据规模的增大, 加速比也会随
之增加.

(3) GPU 程序运行时间的详细分布见表 3.

表 3 HVSBM 并行算法运行时间对比

图片尺寸大小	176 * 144	352 * 288	640 * 480	720 * 576
帧数	300	300	300	300
装载时间(s)	0.022289	0.089112	0.263826	0.294917
GPU 计算时间(s)	0.000012	0.000015	0.000014	0.000012
回传时间(s)	0.033881	0.213804	0.644914	0.716394

实验结果表明:GPU 的计算时间相对于数据从 host 端向 device 段的转移和将结果回传到 host 端的时间,非常小,即 GPU 资源在大部分时间处于空闲状态.应该充分发掘 GPU 的使用率,这有待于进一步研究.

7 结束语

视频图像中块效应评价算法具有密集的计算任务和工作量大的特点,文中通过对相同文件的视频图像块效应串行算法和并行算法运行时间以及正确性的比较,结果表明在 GTX280 上实现的并行算法较 CPU 上的串行算法速度提高了近 30 倍.说明利用 CUDA 技术达到了预期的效果,证明了 CUDA 技术在数字图像处理应用中具有极大的潜力,尤其是计算密集型任务处理.但 GPU 资源的充分使用还有待于进一步的发掘,值得下一步探讨研究.

参考文献:

- [1] Kim T K, Paik J K. Fast image restoration for reducing block artifacts based on adaptive constrained optimization [J]. *Journal of Visual Communication and Image Representation*, 1998, 9(3): 234 - 242.
- [2] Wang Z, Bovik A C. *Modern image quality assessment* [M]. USA, Morgan & Claypool Press, San Rafael, 2006.
- [3] Wu H R, Yuen M. A generalized block - edge impairment metric for video coding [J]. *Signal Processing Letters*, IEEE, 1997, 4(11): 317 - 320.
- [4] Minami S, Zakhor A. An optimization approach for removing blocking effects in transform coding [J]. *Circuits and Systems for Video Technology*, IEEE, 1995, 5(2): 74 - 82.
- [5] Triantafyllidis G A, Tzovaras D. Blocking artifact detection and reduction in compressed data [J]. *Circuits and Systems for Video Technology*, IEEE, 2002, 12(10): 877 - 890.
- [6] ZhaoLin Z, Haoshan S. A novel blind measurement of blocking artifacts for H.264/AVC Video [C]// *Image and Graphics*, Fifth International Conference. Japan, 2009.
- [7] 李政芳,周志恒,曾德炉,等.图像块效应评价标准 [J]. *技术探讨*, 2007(5): 59 - 61
- [8] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture programming guid 2.0 [EB/OL]. (2008 - 06 - 07) [2009 - 12 - 11]. http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf.
- [9] Ying Zhang, Xuejun Yang, Guibin Wang, et al. Scientific computing applications on a stream processor [C]// *Proceedings of IEEE ISPASS*. Austin, Texas, 2008: 105 - 114.
- [10] Owens J D, Houston M, Luebke D, et al. GPU computing [J]. *IEEE*, 2008, 96(5): 879 - 899.

作者简介:

魏晓娟 女, (1985 -), 硕士研究生. 研究方向为并行计算.
陈庆奎 男, (1966 -), 教授, 博士生导师. 研究方向为网络计算、并行理论.
王海峰 男, (1976 -), 博士. 研究方向为并行计算.

(上接第 255 页)

- [7] Zahra Farzanyar, Mohamnadreza Kangavari, Sattar Hashemi. A new algorithm for mining fuzzy association rules in the large databases based on ontology [C]// *Sixth IEEE International Conference on Data Mining - Workshops*, 0 - 7695 - 2702 - 7/06. New York, 2006.
- [8] Claudia Marinica, Fabrice Guillet. Improving post - mining of association rules with ontologies [C]// *The XIII International Conference, Applied Stochastic Models and Data Analysis*. Canada, 2009.
- [9] Joanna Jozefowska, Agnieszka Lawrynowicz, Tomasz Lukaszewski. On reducing redundancy in mining relational association rules from the semantic web [C]// *International Conference on Web Reasoning and Rules Systems*. Germany: karlsruhe, 2008: 205 - 213.
- [10] Gruber T R. A translation approach to portable ontology specifications [J]. *Knowledge Acquisition*, 1993(5): 199 - 220.
- [11] Maedche A. *Ontology learning for the semantic web* [M]. Boston: Kluwer Academic Publishers, 2002.

作者简介:

李帆 男, (1986 -), 硕士研究生. 研究方向为知识发现与数据挖掘.